CS 101: Computer Programming and Utilization

18-Classes

Instructor: Sridhar Iyer
IIT Bombay

What does this program do?

```
struct Date {int day, month, year};

Date make_Date(int d, int m, int y) {
   Date D;
   D.day = d; D.month = m; D.year = y;
   return D;
}

void print(Date D) {
   cout << D.day<<' ';
   switch (D.month) {
   case 1: cout << “Jan”; break;
   … default: “Invalid Month”;
   }
   cout << D.year<<endl; }
```

```
int main(){
Date D = {12, 10, 2012};
int d, m, y;
   cin >> d >> m >> y;
   Date E = make_date(d,m,y);
   print(D);
   print(E);
}
```

struct with member functions

```
struct Date {
int day, month, year;

Date (int d, int m, int y) { //constructor
day = d; month = m; year = y;
}

void print (){
cout << day<<' ';
switch (month) {
case 1: cout << “Jan”; break;
…
}cout << year<<endl;
}
}
```

```
int main() {
  Date D = {12, 10, 2012};
  int d, m, y;
  cin >> d >> m >> y;
  Date E(d,m,y);
  D.print();
  E.print();
}
```

Including functions with data

- Member functions
    - Functions defined inside the struct definition.
- Provide a controlled mechanism using which the struct data can be accessed.
    - Do not allow the user direct access to all the data.
- Helps to separate internal representation of data from operations that can be performed on data
    - Program component = data + functions
    - Program component = internal state + external interface
    - Write programs using external interface without handling internal state directly.

Activity: Implementing a Queue

Suppose you have to write a Taxi Service program.

- When a driver arrives, his ID is entered in an array driverID (if the array has space).
- When a customer arrives the earliest waiting driver (if any) in driverID is assigned to the customer.

Think: What struct and variables are required?

Pair: Discuss the pseudo-code for the functions that are required.

Share: Compare with demo18-queue.cpp

Solution – 1 (without using struct)

```
const int n = 100; // max no of waiting drivers.
int driverID[n], nWaiting = 0, front = 0;
while(true) {
char command; cin >> command;
if(command == 'd') { // driver arrives
    if(nWaiting >= n) cout << "Queue full.\n";
    else{ cin >> driverID[(front + nWaiting) % n]; nWaiting++;
    } //Optimization - use of circular array
}
else if(command == 'c') ...
```

Solution – 2 (using struct + functions)

```
struct Queue {
   int front, nWaiting; int driverID[n]; // n = 100 defined earlier
   Queue() { front=0; nWaiting = 0; }
   bool insert (int value) {
      if(nWaiting == n) return false; // queue is full
      driverID[(front + nWaiting) % n] = value; nWaiting++; return true;
   }
   int remove() {
      if(nWaiting == 0) return -1; // queue is empty
      int driver = driverID[front];
      front = (front + 1) % n; nWaiting--; return driver;
   }  }
```

Solution -2 (contd.)

```
int main() { Queue q;
while(true) {
   char command; cin >> command;
   if(command == 'd') {
      int driver; cin >> driver;
      if (!q.insert(driver)) cout << "Queue full.\n";
   } else if(command == 'c') {
      int driver = q.remove();
      if (driver == -1) cout << "No taxi.\n";
      else cout << "Assigning:" << driver << endl;
   } } }
```

Advantages of solution 2 over solution 1

Observation: In solution 1, statements dealing with what is typed by operator, e.g. command == 'c' etc. mixed with statements dealing with incrementing front, nWaiting etc.

Solution 2 partitions the logic into two parts:

- One part deals with processing commands.
- Another deals with managing front, nWaiting...
- So main program is easier to understand; Queue definition is also easier to understand, not mixed with main program processing.
- Queue class can be used in other programs also

Class = Struct + ?

- Constructor(s)
  - How to set up the initial configuration
  - Perhaps allocate memory
- Destructors
  - Release resources and clean up state
- Methods (functions)
  - Change the state of the struct members in clean, controlled ways
  - Methods are like functions except they have access to the invisible variable “this” (more on this later)
- Access control: public, protected, private
- Inheritance (extending classes)

Why access control?

- “By mistake” someone might write q.front = 0; in the main program.
- The designer of class Queue can decide what parts of the queue are visible to Queue users.

- “public:” : what follows is public, i.e., can be used outside the class definition.
- “private” : what follows is private, i.e., can be used only inside the class definition.
- “protected”: what follows is protected, i.e., can be used only inside the class or “derived” classes (more on this later).

Class: struct + functions + access control

```
class Queue {
private:
int front, nWaiting; // cannot be used outside
int driverID[n]; // cannot be used outside
public:
Queue() { front=0; nWaiting = 0; } // can be called from outside
bool insert(int value) { // can be called from outside
… }
int remove() { // can be called from outside
.... }
};
```

Modified Queue (without circular array)

```
class Queue { // Will work for the same main program!
private: int nWaiting, driverID[n];
public:
Queue() { nWaiting = 0; } // declaration unchanged
bool insert (int value) { // declaration unchanged
  if (nWaiting == n) return false;
  driverID[(nWaiting)] = value; nWaiting++; return true; }
int remove() { // declaration unchanged
  if(nWaiting == 0) return -1; int driver = driverID[0];
  for(int i=1; i < nWaiting; i++) driverID[i-1] = driverID[i];
  nWaiting--; return driver; }};
```

Struct vs. Class

- struct of C: Only data members. No member functions. Data members are public.

- struct of C++ : data members and member functions. Both public by default. You can change this by putting in access control keywords.

- class (only C++): exactly like struct, except that everything inside the definition is private by default. You can change this by putting in access control keywords.